Design of Digital Circuits (252-0028-00L), Spring 2020
Optional HW 6: Vector Processors and GPUs

Instructor: Prof. Onur Mutlu
TAs: Mohammed Alser, Rahul Bera, Can Firtina, Juan Gomez-Luna, Jawad Haj-Yahya, Hasan Hassan, Konstantinos Kanellopoulos, Lois Orosa, Jisung Park, Geraldo De Oliveira Junior, Minesh Patel, Giray Yaglikci

Released: Friday, May 15, 2020

1 Vector Processing I

Consider the following piece of code:

```
for (i = 0; i < 100; i ++)
  A[i] = ((B[i] * C[i]) + D[i])/2;
```

(a) Translate this code into assembly language using the following instructions in the ISA (note the number of cycles each instruction takes is shown next to each instruction):

Opcode	Operands	Number of cycles	Description
LEA	Rd, X	1	Rd ← address of X
LD	Rd, Rs, Rt	11	Rd ← MEM[Rs + Rt]
ST	Rs, Rt, Ru	11	MEM[Rt + Ru] ← Rs
MOVI	Rd, imm	1	Rd ← imm
MUL	Rd, Rs, Rt	6	Rd ← Rs × Rt
ADD	Rd, Rs, Rt	4	Rd ← Rs + Rt
ADD	Rd, Rs, imm	4	Rd ← Rs + imm
RSHFA	Rd, Rs, shamt	1	Rd ← Rs >>> shamt
BR<CC>	Rs, X	1	Branch to X if Rs satisfies condition code CC

Assume one memory location is required to store each element of the array. Also assume that there are eight register, R0 to R7.

Condition codes are set after the execution of an arithmetic instruction. You can assume typically available condition codes such as zero (EQZ), positive (GTZ), negative (LTZ), non-negative (GEZ), and non-positive (LEZ).

How many cycles does it take to execute the program?

(b) Now write Cray-like vector assembly code to perform this operation in the shortest time possible. Assume that there are eight vector registers and the length of each vector register is 64. Use the following instructions in the vector ISA:

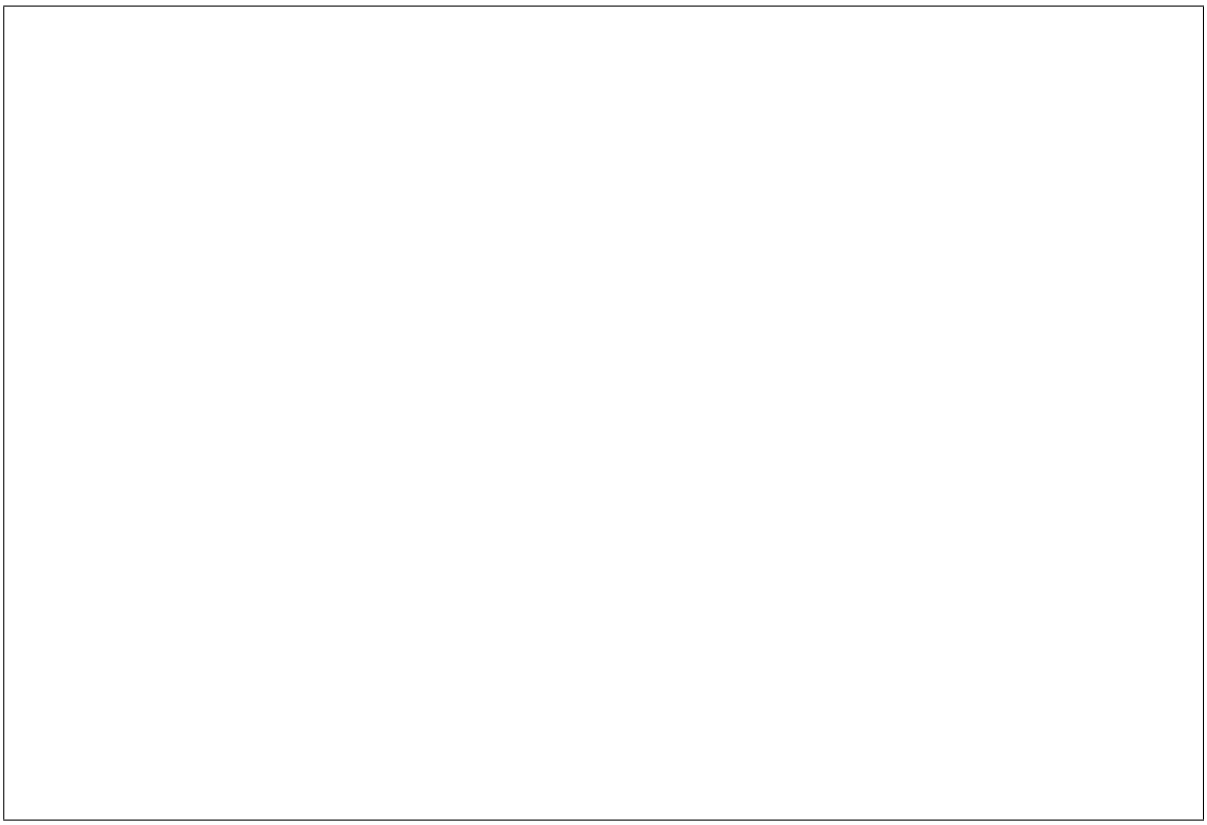
Opcode	Operands	Number of cycles	Description
SET	Vst, imm	1	Vst ← imm (Vst: Vector Stride Register)
SET	Vln, imm	1	Vln ← imm (Vln: Vector Length Register)
VLD	Vd, X	11, pipelined	Vd ← MEM[address of X]
VST	Vs, X	11, pipelined	MEM[address of X] ← Vs
VADD	Vd, Vs, Vt	4, pipelined	Vd ← Vs + Vt
VMUL	Vd, Vs, Vt	6, pipelined	$Vd_i \leftarrow Vs_i \times Vt_i$
VRSHFA	Vd, Vs, shamt	1	$Vd_i \leftarrow Vs_i >>> shamt$

How many cycles does it take to execute the program on the following processors? Assume that memory is 16-way interleaved.

Vector processor without chaining, 1 port to memory (1 load or store per cycle):

Vector processor with chaining, 1 port to memory:

Vector processor with chaining, 2 read ports and 1 write port to memory:



2 Vector Processing II

You are studying a program that runs on a vector computer with the following latencies for various instructions:

- `VLD` and `VST`: 50 cycles for each vector element; fully interleaved and pipelined.
- `VADD`: 4 cycles for each vector element (fully pipelined).
- `VMUL`: 16 cycles for each vector element (fully pipelined).
- `VDIV`: 32 cycles for each vector element (fully pipelined).
- `VRSHFA`: 1 cycle for each vector element (fully pipelined).

Assume that:

- The machine has an in-order pipeline.
- The machine supports chaining between vector functional units.
- In order to support 1-cycle memory access after the first element in a vector, the machine interleaves vector elements across memory banks. All vectors are stored in memory with the first element mapped to bank 0, the second element mapped to bank 1, and so on.
- Each memory bank has an 8 KB row buffer.
- Vector elements are 64 bits in size.
- Each memory bank has two ports (so that two loads/stores can be active simultaneously), and there are two load/store functional units available.

(a) What is the minimum power-of-two number of banks required in order for memory accesses to never stall? (Assume a vector stride of 1.)

(b) The machine (with as many banks as you found in part a) executes the following program (assume that the vector stride is set to 1):

```
VLD    V1, A         // V1  ← A
VLD    V2, B         // V2  ← B
VADD   V3, V1, V2    // V3  ← V1 + V2
VMUL   V4, V3, V1    // V4_i ← V3_i × V1_i
VRSHFA V5, V4, 2     // V5_i ← V4_i >>> 2
```

It takes 111 cycles to execute this program. What is the vector length L (i.e., the number of elements in a vector)?

If the machine did not support chaining (but could still pipeline independent operations), how many cycles would be required to execute the same program?

(c) The architect of this machine decides that she needs to cut costs in the machine's memory system. She reduces the number of banks by a factor of 2 from the number of banks you found in part (a) above. Because loads and stores might stall due to bank contention, an *arbiter* is added to each bank so that pending loads from the oldest instruction are serviced first. How many cycles does the program take to execute on the machine with this reduced-cost memory system (but with chaining)?

Now, the architect reduces cost further by reducing the number of memory banks (to a lower power of 2). The program executes in 279 cycles. How many banks are in the system?

(d) Another architect is now designing the second generation of this vector computer. He wants to build a multicore machine in which 4 vector processors share the same memory system. He scales up the number of banks by 4 in order to match the memory system bandwidth to the new demand. However, when he simulates this new machine design with a separate vector program running on every core, he finds that the average execution time is longer than if each individual program ran on the original single-core system with 1/4 the banks. Why could this be? Provide concrete reason(s).

What change could this architect make to the system in order to alleviate this problem (in less than 20 words), while *only* changing the shared memory hierarchy?

3 Vector Processing III

Assume a vector processor that implements the following ISA:

Opcode	Operands	Number of cycles	Description
SET	Vst, imm	1	Vst ← imm (Vst: Vector Stride Register)
SET	Vln, imm	1	Vln ← imm (Vln: Vector Length Register)
VLD	Vd, addr	100, pipelined	Vd ← MEM[addr]
VST	Vs, addr	100, pipelined	MEM[addr] ← Vs
VADD	Vd, Vs, Vt	5, pipelined	Vd ← Vs + Vt
VMUL	Vd, Vs, Vt	10, pipelined	$Vd_i \leftarrow Vs_i \times Vt_i$
VDIV	Vd, Vs, Vt	20, pipelined	$Vd_i \leftarrow Vs_i \ / \ Vt_i$

Assume the following:

- The processor has an in-order pipeline.
- The size of a vector element is 4 bytes.
- Vst and Vln are 10-bit registers.
- The processor *does not* support chaining between vector functional units.
- The main memory has N banks.
- Vector elements stored in consecutive memory addresses are interleaved between the memory banks. For example, if a vector element at address A maps to bank B, a vector element at address $A + 4$ maps to bank $(B + 1)\%N$, where % is the modulo operator and N is the number of banks. N is *not necessarily* a power of two.
- The memory is byte addressable and the address space is represented using 32 bits.
- Vector elements are stored in memory in 4-byte-aligned manner.
- Each memory bank has a 4 KB row buffer.
- Each memory bank has a single read and a single write port so that a load and a store operation can be performed simultaneously.
- There are separate functional units for executing VLD and VST instructions.

(a) What should the minimum value of N be to avoid stalls while executing a VLD or VST instruction, assuming a vector stride of 1? Explain.

(b) What should the minimum value of N be to avoid stalls while executing a `VLD` or `VST` instruction, assuming a vector stride of 2? Explain.

(c) Assume:

- A machine that has a memory with as many banks as you found is part (a).
- The vector stride is set to 1.
- The value of the vector length is set to M (but we do *not* know M)

The machine executes the following program:

```
VLD  V1, A              // V1 ← MEM[A]
VLD  V2, (A + 32768)    // V2 ← MEM[A + 32768]
VADD V3, V1, V1         // V3 ← V1, V1
VMUL V4, V2, V3         // V4_i ← V2_i, V3_i
VST  V4, (A + 32768*2)  // MEM[A + 32768*2] ← V4
```

It takes 4,306 cycles to execute the above program. What is M? Explain.

(d) If we modify the vector processor to *support chaining*, how many cycles would be required to execute the same program in part (c)? Explain.

4 SIMD Processing

Suppose we want to design a SIMD engine that can support a vector length of 16. We have two options: a traditional vector processor and a traditional array processor.

(a) Which one is more costly in terms of chip area (circle one)?

`The traditional vector processor` `The traditional array processor` `Neither`

Justify your answer.

(b) Assuming the latency of an addition operation is five cycles in both processors, how long will a `VADD` (vector add) instruction take in each of the processors (assume that the adder can be fully pipelined and is the same for both processors)?

For a vector length of 1:

The traditional vector processor:

The traditional array processor:

For a vector length of 4:

The traditional vector processor:

The traditional array processor:

For a vector length of 16:

The traditional vector processor:

The traditional array processor:

5 GPUs and SIMD I

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A and B are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 2 instructions in each thread.) A warp in the GPU consists of 32 threads, there are 32 SIMD lanes in the GPU. Assume that each instruction takes the same amount of time to execute.

```
for (i = 0; i < N; i++) {
    if (A[i] % 3 == 0) {     // Instruction 1
       A[i] = A[i] * B[i];   // Instruction 2
    }
}
```

(a) How many warps does it take to execute this program? Please leave the answer in terms of N.

(b) Assume integer arrays A have a repetitive pattern which have 24 ones followed by 8 zeros repetitively and integer arrays B have a different repetitive pattern which have 48 zeros followed by 64 ones. What is the SIMD utilization of this program?

(c) Is it possible for this program to yield a SIMD utilization of 100%? Circle one.

`YES` `NO`

If `YES`, what should be true about array `A` for the SIMD utilization to be 100%?

What should be true about array `B`?

If `NO`, explain why not.

(d) Is it possible for this program to yield a SIMD utilization of 56.25%? Circle one.

`YES` `NO`

If `YES`, what should be true about array `A` for the SIMD utilization to be 56.25%?

What should be true about array `B`?

If `NO`, explain why not.

(e) Is it possible for this program to yield a SIMD utilization of 50%? Circle one.

`YES` `NO`

If `YES`, what should be true about array `A` for the SIMD utilization to be 50%?

What should be true about array `B`?

If NO, explain why not.

Now, we will look at the technique we learned in class that tries to improve SIMD utilization by merging divergent branches together. The key idea of the *dynamic warp formation* is that threads in one warp can be swapped with threads in another warp as long as the swapped threads have access to the associated registers (i.e., they are on the same SIMD lane).

Consider the following example of a program that consists of 3 warps X, Y and Z that are executing the same code segment specified at the top of this question. Assume that the vector below specifies the direction of the branch of each thread within the warp. 1 means the branch in Instruction 1 is resolved to taken and 0 means the branch in Instruction 1 is resolved to not taken.

```
X = {10000000000000000000000000000010}
Y = {10000000000000000000000000000001}
Z = {01000000000000000000000000000000}
```

(f) Given the example above. Suppose that you perform dynamic warp formation on these three warps. What is the resulting outcome of each branch for the newly formed warps X', Y' and Z'.

(g) Given the specification for arrays A and B, is it possible for this program to yield a better SIMD utilization if dynamic warp formation is used? Explain your reasoning.

6 GPUs and SIMD II

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program. As we saw in lecture and practice exercises, the SIMD utilization of a program is computed across the *complete run* of the program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays `A`, `B`, and `C` are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 6 instructions in each thread.) A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU. Please assume that all values in array `B` have magnitudes less than 10 (i.e., $|\texttt{B[i]}| < 10$, for all `i`).

```
for (i = 0; i < 1024; i++) {
    A[i] = B[i] * B[i];
    if (A[i] > 0) {
        C[i] = A[i] * B[i];
        if (C[i] < 0) {
            A[i] = A[i] + 1;
        }
        A[i] = A[i] - 2;
    }
}
```

(a) How many warps does it take to execute this program?

(b) What is the maximum possible SIMD utilization of this program?

(c) Please describe what needs to be true about array `B` to reach the maximum possible SIMD utilization asked in part (b). (Please cover all cases in your answer)

(d) What is the minimum possible SIMD utilization of this program?

(e) Please describe what needs to be true about array `B` to reach the minimum possible SIMD utilization asked in part (d). (Please cover all cases in your answer)

7 GPUs and SIMD III

We define the *SIMD utilization* of a program that runs on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of the program. As we saw in lecture and practice exercises, the SIMD utilization of a program is computed across the *complete run* of the program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays `A` and `B` are already in vector registers, so there are no loads and stores in this program. (Hint: Notice that there are 3 instructions in each iteration.) A warp in the GPU consists of 32 threads, and there are 32 SIMD lanes in the GPU.

```
for (i = 0; i < 1025; i++) {
    if (A[i] < 33) {        // Instruction 1
        B[i] = A[i] << 1;   // Instruction 2
    }
    else {
        B[i] = A[i] >> 1;   // Instruction 3
    }
}
```

Please answer the following six questions.

(a) How many warps does it take to execute this program?

(b) What is the *maximum* possible SIMD utilization of this program? (Hint: The warp scheduler does *not* issue instructions when *no* threads are active).

(c) Please describe what needs to be true about array `A` to reach the maximum possible SIMD utilization asked in part (b). (Please cover all cases in your answer.)

(d) What is the *minimum* possible SIMD utilization of this program?

(e) Please describe what needs to be true about array `A` to reach the minimum possible SIMD utilization asked in part (d). (Please cover all cases in your answer.)

(f) What is the SIMD utilization of this program if `A[i] = i`? Show your work.

8 GPUs and SIMD IV

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A, B, and C are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 6 instructions in each thread.) A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU.

```
for (i = 0; i < 4096; i++) {
    if (B[i] < 8888) {
        A[i] = A[i] * C[i];
        A[i] = A[i] + B[i];
        C[i] = B[i] + 1;
    }
    if (B[i] > 8888) {
        A[i] = A[i] * B[i];
    }
}
```

(a) How many warps does it take to execute this program?

(b) When we measure the SIMD utilization for this program with one input set, we find that it is 134/320. What can you say about arrays A, B, and C? Be precise (Hint: Look at the "if" branch).

A:

B:

C:

(c) Is it possible for this program to yield a SIMD utilization of 100% (circle one)?

YES NO

If YES, what should be true about arrays A, B, C for the SIMD utilization to be 100%? Be precise. If NO, explain why not.

(d) What is the lowest SIMD utilization that this program can yield? Explain.